# Spring中单元测试

@ContextConfiguration(locations = {

"classpath:xxx.xml","classpath:xxxx.xml"})

**public** **class** ClazzTest **extends** AbstractJUnit4SpringContextTests

--------------------------------------------------------------------------------------------------------------------

# Spring中应用属性文件读入

-----------------------------------------------------------------

**applicationContext.xml**

<bean id=*"propertyConfigurer"*

class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"systemPropertiesModeName"* value=*"SYSTEM\_PROPERTIES\_MODE\_OVERRIDE"*/>

<property name=*"properties"* ref=*"applicationProperties"*/>

</bean>

<bean id=*"applicationProperties"*

class=*"org.springframework.beans.factory.config.PropertiesFactoryBean"*>

<property name=*"ignoreResourceNotFound"* value=*"true"*/>

<property name=*"locations"*>

<list>

<value>classpath:web.properties</value>

</list>

</property>

<property name=*"fileEncoding"* value=*"UTF-8"*/>

</bean>

<bean id=*"testPO"* class=*"example.po.TestPO"*>

<property name=*"name"* value=*"${my.name}"*/>

</bean>

-----------------------------------------------------------------

**Confing.properties**

url=www.baidu.com

my.name=xiaoming

-----------------------------------------------------------------

**ConfigInfo.java**

@Component("configInfo")

**public** **class** ConfigInfo {

@Value("#{url}")

**private** String url;

**public** String getUrl() {

**return** url;

}

**public** **void** setUrl(String url) {

**this**.url = url;

}

}

--------------------------------------------------------------------------------------------------------------------

# Spring中获取上下文(ApplicationContext)

-----------------------------------------------------------------

**MyContextLoaderListener.java**

**public** **class** MyContextLoaderListener **extends** ContextLoaderListener{

@Override

**public** **void** contextInitialized(ServletContextEvent event) {

**super**.contextInitialized(event);

//初始化SpringContextUtil的context

ServletContext context = event.getServletContext();

ApplicationContext ctx = WebApplicationContextUtils.getRequiredWebApplicationContext(context);

ConfigInfo configInfo = (ConfigInfo) ctx.getBean("configInfo");

System.out.println(configInfo.getUrl());

TestPO testPO = (TestPO) ctx.getBean("testPO");

System.out.println(testPO.getName());

}

}

-----------------------------------------------------------------

**web.xml**

<listener>

<listener-class>example.listener.MyContextLoaderListener</listener-class>

</listener>

--------------------------------------------------------------------------------------------------------------------

# request中获取ServletContext

-----------------------------------------------------------------

HttpServletRequest request = (HttpServletRequest) req;

ServletContext application = request.getSession().getServletContext();

--------------------------------------------------------------------------------------------------------------------

# AOP切面(日志)

-----------------------------------------------------------------

**ActionLogAspect.java**

@Component

@Aspect

**public** **class** ActionLogAspect {

**private** **static** **final** Logger logger = LoggerFactory.getLogger(ActionLogAspect.**class**);

//配置切入点

@Pointcut("execution(public \* com.damuzee.web.app.\*.\*(..))" + // 配置app切入点

"||execution(public \* com.damuzee.web.common.\*.\*(..))" + // 配置commom切入点

"||execution(public \* com.damuzee.web.wechat.\*.\*(..))" + // 配置wechat切入点

"||execution(public \* com.damuzee.web.weixin.\*.\*(..))" + // 配置wechat切入点

**public** **void** pointcut(){}

@Around("pointcut()")

**public** Object aroundMethod(ProceedingJoinPoint pjd) {

//执行目标方法

String methodName = pjd.getSignature().getName();

Object result = **null**;

**try** {

//前置通知

methodName = pjd.getStaticPart().toString();

methodName = methodName.substring(methodName.lastIndexOf('.') + 1, methodName.lastIndexOf(')'));

logger.info("执行{}方法开始", methodName);

logger.info("传入{}方法参数为：{}", methodName, pjd.getArgs());

logger.info("{}方法所属类为{}", methodName, pjd.getTarget().getClass());

result = pjd.proceed();

//返回通知

**return** result;

} **catch** (ActionException e) {

**throw** e;

} **catch** (ServiceException e) {

**throw** **new** ActionException(e.getMessage(), e);

}**catch** (Throwable e) {

**throw** ActionException.instance(e);

} **finally** {

//后置通知

logger.info("执行{}方法结束, 输出结果为：{}", methodName, result);

}

}

}

-----------------------------------------------------------------

**applictaionContext.xml**

<!-- 启动AspectJ支持 “日志通知”-->

<aop:aspectj-autoproxy />

--------------------------------------------------------------------------------------------------------------------

# 文件复制

-----------------------------------------------------------------

**FileUtils.java**

**public** **class** FileUtils {

**public** **static** **void** copyFile(File sourcefile, File targetFile) {

FileInputStream fi = **null**;

FileOutputStream fo = **null**;

FileChannel in = **null**;

FileChannel out = **null**;

**try** {

fi = **new** FileInputStream(sourcefile);

fo = **new** FileOutputStream(targetFile);

in = fi.getChannel();//得到对应的文件通道

out = fo.getChannel();//得到对应的文件通道

in.transferTo(0, in.size(), out);//连接两个通道，并且从in通道读取，然后写入out通道

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

fi.close();

in.close();

fo.close();

out.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

--------------------------------------------------------------------------------------------------------------------

# 代理（JDK）

-----------------------------------------------------------------

**Subject.java**

**public** **interface** Subject {

**abstract** **public** **void** request();

}

-----------------------------------------------------------------

**RealSubject.java**

**public** **class** RealSubject **implements** Subject {

**public** RealSubject() {}

**public** **void** request() {

System.out.println( " From real subject. " );

}

}

-----------------------------------------------------------------

**DynamicSubject.java**

**public** **class** DynamicSubject **implements** InvocationHandler {

**private** Object sub;

**public** DynamicSubject() {

}

**public** DynamicSubject(Object obj) {

sub = obj;

}

**public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {

System.out.println(" before calling " + method);

method.invoke(sub, args);

System.out.println(" after calling " + method);

**return** **null**;

}

}

-----------------------------------------------------------------

**Test.java**

**public** **class** Test {

**static** **public** **void** main(String[] args) **throws** Throwable {

RealSubject rs = **new** RealSubject(); // 在这里指定被代理类

InvocationHandler ds = **new** DynamicSubject(rs);

Class cls = rs.getClass();

// 以下是一次性生成代理

Subject subject = (Subject) Proxy.newProxyInstance(cls.getClassLoader(), cls.getInterfaces(), ds);

subject.request();

}

}

--------------------------------------------------------------------------------------------------------------------

# 代理（CGLIB）

-----------------------------------------------------------------

**BookFacade.java**

**public** **class** BookFacade {

**public** **void** addBook() {

System.out.println("增加图书的普通方法...");

}

}

-----------------------------------------------------------------

**BookFacadeCglib.java**

**public** **class** BookFacadeCglib **implements** MethodInterceptor {

**private** Object target;

/\*\*

\* 创建代理对象

\*/

**public** Object getInstance(Object target) {

**this**.target = target;

Enhancer enhancer = **new** Enhancer();

enhancer.setSuperclass(**this**.target.getClass());

// 回调方法

enhancer.setCallback(**this**);

// 创建代理对象

**return** enhancer.create();

}

@Override

// 回调方法

**public** Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy) **throws** Throwable {

System.out.println("事物开始");

proxy.invokeSuper(obj, args);

System.out.println("事物结束");

**return** **null**;

}

}

-----------------------------------------------------------------

**Test.java**

**public** **class** Test {

**public** **static** **void** main(String[] args) {

BookFacadeCglib cglib=**new** BookFacadeCglib();

BookFacadeImpl bookCglib=(BookFacadeImpl)cglib.getInstance(**new** BookFacadeImpl());

bookCglib.addBook();

}

}

--------------------------------------------------------------------------------------------------------------------

# 邮件发送

-----------------------------------------------------------------

**SendMailUtils**

**import** java.util.Properties;

**import** javax.mail.Address;

**import** javax.mail.Message;

**import** javax.mail.Session;

**import** javax.mail.Transport;

**import** javax.mail.internet.InternetAddress;

**import** javax.mail.internet.MimeMessage;

**public** **class** SendMailUtils {

/\*\*

\* **@param** title 邮件标题

\* **@param** content 邮件内容

\* **@param** emilAddress 收件人地址

\* **@exception** Exception

\*/

**public** **static** **void** sendEmail(String sendEmail, String sendEmailPwd, String title, String content, String[] toEmilAddress) **throws** Exception {

Properties props = **new** Properties();

props.setProperty("mail.smtp.auth", "true");

props.setProperty("mail.transport.protocol", "smtp");

props.setProperty("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

Session session = Session.getInstance(props);

Message msg = **new** MimeMessage(session);

// 发送的邮箱地址

msg.setFrom(**new** InternetAddress(sendEmail));

// 设置标题

msg.setSubject(title);

// 设置内容

msg.setContent(content, "text/html;charset=gbk;");

Transport transport = session.getTransport();

// 设置服务器以及账号和密码

transport.connect("smtp.exmail.qq.com", 465, sendEmail, sendEmailPwd);

// 发送到的邮箱地址

transport.sendMessage(msg, getAddress(toEmilAddress));

transport.close();

}

**private** **static** Address[] getAddress(String[] emilAddress) **throws** Exception {

Address[] address = **new** Address[emilAddress.length];

**for** (**int** i = 0; i < address.length; i++) {

address[i] = **new** InternetAddress(emilAddress[i]);

}

**return** address;

}

}

--------------------------------------------------------------------------------------------------------------------

# 验证码

-----------------------------------------------------------------

**UserController.java**

@RequestMapping(value = "checkCode")

**public** **void** checkCode(HttpServletRequest request, HttpServletResponse response, HttpSession session) **throws**

IOException {

CreateImageCode vCode = **new** CreateImageCode(150, 33, 5, 10);

response.setHeader("Pragma", "no-cache");

response.setHeader("Cache-Control", "no-cache");

response.setDateHeader("Expires", 0);

response.setContentType("image/jpeg");

String code = vCode.getCode();

session.setAttribute(Constants.CHECK\_CODE\_KEY, code);

vCode.write(response.getOutputStream());

}

-----------------------------------------------------------------

**CreateImageCode.java**

**public** **class** CreateImageCode {

// 图片的宽度。

**private** **int** width = 160;

// 图片的高度。

**private** **int** height = 40;

// 验证码字符个数

**private** **int** codeCount = 4;

// 验证码干扰线数

**private** **int** lineCount = 20;

// 验证码

**private** String code = **null**;

// 验证码图片Buffer

**private** BufferedImage buffImg = **null**;

Random random = **new** Random();

**public** CreateImageCode() {

creatImage();

}

**public** CreateImageCode(**int** width, **int** height) {

**this**.width = width;

**this**.height = height;

creatImage();

}

**public** CreateImageCode(**int** width, **int** height, **int** codeCount) {

**this**.width = width;

**this**.height = height;

**this**.codeCount = codeCount;

creatImage();

}

**public** CreateImageCode(**int** width, **int** height, **int** codeCount, **int** lineCount) {

**this**.width = width;

**this**.height = height;

**this**.codeCount = codeCount;

**this**.lineCount = lineCount;

creatImage();

}

// 生成图片

**private** **void** creatImage() {

**int** fontWidth = width / codeCount;// 字体的宽度

**int** fontHeight = height - 5;// 字体的高度

**int** codeY = height - 8;

// 图像buffer

buffImg = **new** BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics g = buffImg.getGraphics();

//Graphics2D g = buffImg.createGraphics();

// 设置背景色

g.setColor(getRandColor(200, 250));

g.fillRect(0, 0, width, height);

// 设置字体

//Font font1 = getFont(fontHeight);

Font font = **new** Font("Fixedsys", Font.BOLD, fontHeight);

g.setFont(font);

// 设置干扰线

**for** (**int** i = 0; i < lineCount; i++) {

**int** xs = random.nextInt(width);

**int** ys = random.nextInt(height);

**int** xe = xs + random.nextInt(width);

**int** ye = ys + random.nextInt(height);

g.setColor(getRandColor(1, 255));

g.drawLine(xs, ys, xe, ye);

}

// 添加噪点

**float** yawpRate = 0.01f;// 噪声率

**int** area = (**int**) (yawpRate \* width \* height);

**for** (**int** i = 0; i < area; i++) {

**int** x = random.nextInt(width);

**int** y = random.nextInt(height);

buffImg.setRGB(x, y, random.nextInt(255));

}

String str1 = randomStr(codeCount);// 得到随机字符

**this**.code = str1;

**for** (**int** i = 0; i < codeCount; i++) {

String strRand = str1.substring(i, i + 1);

g.setColor(getRandColor(1, 255));

// g.drawString(a,x,y);

// a为要画出来的东西，x和y表示要画的东西最左侧字符的基线位于此图形上下文坐标系的 (x, y) 位置处

g.drawString(strRand, i \* fontWidth + 3, codeY);

}

}

// 得到随机字符

**private** String randomStr(**int** n) {

String str1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890";

String str2 = "";

**int** len = str1.length() - 1;

**double** r;

**for** (**int** i = 0; i < n; i++) {

r = (Math.random()) \* len;

str2 = str2 + str1.charAt((**int**) r);

}

**return** str2;

}

// 得到随机颜色

**private** Color getRandColor(**int** fc, **int** bc) {// 给定范围获得随机颜色

**if** (fc > 255) fc = 255;

**if** (bc > 255) bc = 255;

**int** r = fc + random.nextInt(bc - fc);

**int** g = fc + random.nextInt(bc - fc);

**int** b = fc + random.nextInt(bc - fc);

**return** **new** Color(r, g, b);

}

/\*\*

\* 产生随机字体

\*/

**private** Font getFont(**int** size) {

Random random = **new** Random();

Font font[] = **new** Font[5];

font[0] = **new** Font("Ravie", Font.PLAIN, size);

font[1] = **new** Font("Antique Olive Compact", Font.PLAIN, size);

font[2] = **new** Font("Fixedsys", Font.PLAIN, size);

font[3] = **new** Font("Wide Latin", Font.PLAIN, size);

font[4] = **new** Font("Gill Sans Ultra Bold", Font.PLAIN, size);

**return** font[random.nextInt(5)];

}

// 扭曲方法

**private** **void** shear(Graphics g, **int** w1, **int** h1, Color color) {

shearX(g, w1, h1, color);

shearY(g, w1, h1, color);

}

**private** **void** shearX(Graphics g, **int** w1, **int** h1, Color color) {

**int** period = random.nextInt(2);

**boolean** borderGap = **true**;

**int** frames = 1;

**int** phase = random.nextInt(2);

**for** (**int** i = 0; i < h1; i++) {

**double** d = (**double**) (period >> 1) \* Math.sin((**double**) i / (**double**) period + (6.2831853071795862D \* (**double**) phase) / (**double**) frames);

g.copyArea(0, i, w1, 1, (**int**) d, 0);

**if** (borderGap) {

g.setColor(color);

g.drawLine((**int**) d, i, 0, i);

g.drawLine((**int**) d + w1, i, w1, i);

}

}

}

**private** **void** shearY(Graphics g, **int** w1, **int** h1, Color color) {

**int** period = random.nextInt(40) + 10; // 50;

**boolean** borderGap = **true**;

**int** frames = 20;

**int** phase = 7;

**for** (**int** i = 0; i < w1; i++) {

**double** d = (**double**) (period >> 1) \* Math.sin((**double**) i / (**double**) period + (6.2831853071795862D \* (**double**) phase) / (**double**) frames);

g.copyArea(i, 0, 1, h1, 0, (**int**) d);

**if** (borderGap) {

g.setColor(color);

g.drawLine(i, (**int**) d, i, 0);

g.drawLine(i, (**int**) d + h1, i, h1);

}

}

}

**public** **void** write(OutputStream sos) **throws** IOException {

ImageIO.write(buffImg, "png", sos);

sos.close();

}

**public** BufferedImage getBuffImg() {

**return** buffImg;

}

**public** String getCode() {

**return** code.toLowerCase();

}

}

--------------------------------------------------------------------------------------------------------------------

# 登录（cookie）

-----------------------------------------------------------------

**UserController.java**

@ResponseBody

@RequestMapping(value = "login.do")

**public** AjaxResponse<Object> loginDo(HttpSession session, HttpServletRequest request, HttpServletResponse

response, String account, String password, String rememberMe, String checkCode) {

**final** String REMEMBERME = "1";

String jsessionId = session.getId();

logger.info("请求的sessionId:{}", jsessionId);

AjaxResponse<Object> result = **new** AjaxResponse<Object>();

result.setResponseCode(ResponseCode.SUCCESS);

**try** {

String sessionCheckCode = String.valueOf(session.getAttribute(Constants.CHECK\_CODE\_KEY));

**if** ((!StringTools.isEmpty(sessionCheckCode) && !sessionCheckCode.equalsIgnoreCase(checkCode)) && **null** !=

session.getAttribute(Constants.SESSION\_ERROR\_LOGIN\_COUNT) && (Integer) session.getAttribute

(Constants.SESSION\_ERROR\_LOGIN\_COUNT) >= Constants.MAX\_LOGIN\_ERROR\_COUNT) {

result.setErrorMsg("验证码错误");

result.setResponseCode(ResponseCode.CODEERROR);

**return** result;

}

User user = userService.login(account, password, **true**, **this**.getIpAddr(request));

SessionUser sessionUser = **new** SessionUser();

sessionUser.setUserId(user.getUserId());

sessionUser.setUserIcon(user.getUserIcon());

sessionUser.setUserName(user.getUserName());

sessionUser.setEditorType(user.getEditorType());

session.setAttribute(Constants.SESSION\_USER\_KEY, sessionUser);

//记住登陆状态

**if** (REMEMBERME.equals(rememberMe)) {

// 自动登录，保存用户名密码到 Cookie

String infor = URLEncoder.encode(account.toString(), "utf-8") + "|" + user.getPassword();

// 清除之前的Cookie 信息

Cookie cookie = **new** Cookie(Constants.COOKIE\_USER\_INFO, **null**);

cookie.setPath("/");

cookie.setMaxAge(0);

// 建用户信息保存到Cookie中

cookie = **new** Cookie(Constants.COOKIE\_USER\_INFO, infor);

cookie.setPath("/");

// 设置最大生命周期为1年。

cookie.setMaxAge(31536000);

response.addCookie(cookie);

} **else** {

Cookie cookie = **new** Cookie(Constants.COOKIE\_USER\_INFO, **null**);

cookie.setMaxAge(0);

cookie.setPath("/");

response.addCookie(cookie);

}

} **catch** (BusinessException e) {

**if** (**null** == session.getAttribute(Constants.SESSION\_ERROR\_LOGIN\_COUNT)) {

session.setAttribute(Constants.SESSION\_ERROR\_LOGIN\_COUNT, 1);

} **else** {

session.setAttribute(Constants.SESSION\_ERROR\_LOGIN\_COUNT, (Integer) session.getAttribute(Constants

.SESSION\_ERROR\_LOGIN\_COUNT) + 1);

}

**if** ((Integer) session.getAttribute(Constants.SESSION\_ERROR\_LOGIN\_COUNT) >= Constants

.MAX\_LOGIN\_ERROR\_COUNT) {

result.setResponseCode(ResponseCode.MOREMAXLOGINCOUNT);

} **else** {

result.setResponseCode(ResponseCode.BUSINESSERROR);

}

result.setErrorMsg(e.getMessage());

logger.error("登陆失败，账号：{}", account, e);

} **catch** (Exception e) {

result.setErrorMsg(ResponseCode.SERVERERROR.getDesc());

result.setResponseCode(ResponseCode.SERVERERROR);

logger.error("登陆失败，账号：{}", account, e);

}

**return** result;

}

-----------------------------------------------------------------

**index.html**

<#if Session["session\_user\_key"]??>

<div class=*"login"* id=*"login-info"*>

<span class=*"user-icon"*><img

src=*"${absolutePath}/resource/images/defusericon/usericon1.png"*></span>

<span class=*"welcome"*>欢迎回来，</span><a

href=*"${absolutePath}/user/${session\_user\_key.userId}"*>${session\_user\_key.userName}</a>

</div>

<div id=*"user-info-dialog"*>

<div>

<div class=*"d-user-icon"*>

<a href=*"${absolutePath}/user/${session\_user\_key.userId}"*><img

src=*"${imageDomain}upload/${session\_user\_key.userIcon}"*></a>

</div>

<div class=*"d-user-info"*>

<div class=*"d-user-info-t"*>${session\_user\_key.userName}</div>

<div class=*"d-user-info-t"*>

关注：<span id=*"focus-count"* class=*"user-count-info"*></span>

</div>

<div class=*"d-user-info-t"*>

粉丝：<span id=*"fans-count"* class=*"user-count-info"*></span>

</div>

<div class=*"d-user-info-t"*>

积分：<span id=*"mark-count"* class=*"user-count-info"*></span>

</div>

<div class=*"d-user-info-t"*>

<a href=*"javascript:;"* class=*"btn small-btn"* id=*"logout-btn"*>退出</a>

<a href=*"${absolutePath}/admin/update\_user.action"*

class=*"btn small-btn"*>后台管理</a>

</div>

</div>

<div class=*"clear"*></div>

</div>

<div class=*"d-title"*>个人设置</div>

<div>

<a href=*"${absolutePath}/user/${session\_user\_key.userId}"*

class=*"info-d"*>个人中心</a> <a

href=*"${absolutePath}/admin/message\_list.action"* class=*"info-d"*>我的消息</a>

<a href=*"${absolutePath}/admin/update\_user.action"* class=*"info-d"*>修改个人信息</a>

<a href=*"${absolutePath}/admin/change\_user\_icon.action"* class=*"info-d"*>修改头像</a>

<a href=*"${absolutePath}/admin/add\_blog.action"* class=*"info-d"*>写博客</a>

<a href=*"${absolutePath}/admin/collection\_list.action?articleType=T"*

class=*"info-d"*>我的收藏</a>

</div>

</div>

<#else>

<div class=*"no-login"*>

<span class=*"def-icon"*><img

src=*"${absolutePath}/resource/images/defusericon/usericon1.png"*></span>

<a href=*"${absolutePath}/login"*>登录</a><span class=*"line"*>|</span><a

href=*"${absolutePath}/register"*>加入有乐窝</a>

</div>

</#if>

--------------------------------------------------------------------------------------------------------------------

# Solr

-----------------------------------------------------------------

**安装：http://blog.csdn.net/liuzhen917/article/details/70328214?utm\_source=itdadao&utm\_medium=referral**

环境介绍

solr 6.5

tomcat8

jdk1.8

win7系统

-----------------------------------------------------------------

**一、下载solr安装包**

下载地址：[http://www.apache.org/dyn/closer.lua/lucene/solr/6.5.0](http://www.apache.org/dyn/closer.lua/lucene/solr/6.5.0" \t "http://blog.csdn.net/liuzhen917/article/details/_blank)

-----------------------------------------------------------------

**二、安装solr**

以win7为例

1、将下载下来的solr-6.5.0.zip解压

2、在D:\work\solr\_root\下新建文件夹solr\_home

3、在solr\_home中新建文件夹solr\_core

4、从solr-6.5.0\solr\server\solr下的三个文件copy到solr\_home中

5、将\solr-6.5.0\solr\server\solr\configsets\basic\_configs下的conf文件夹copy到solr\_core中

-----------------------------------------------------------------

**三、solr应用部署到tomcat下**

1、将\solr-6.5.0\server\solr-webapp中的web复制到tomcat的webapp下，并将web名字修改为solr

2、添加需要的扩展依赖包

从\solr-6.5.0\server\lib\ext下将所有的jar包复制到tomcat的webapps下的solr项目的lib目录中

复制solr-6.5.0\solr-6.5.0\server\lib下的metrics-相关的jar包到webapps下的solr项目中

3、添加log4j的配置文件

从\solr-6.5.0\server\resources 下将log4j配置文件复制到tomcat的webapps下的classes文件夹中，classes

4、配置web.xml指定solrhome的位置

打开tomcat下的solr的web.xml，将下图中的部分去掉注释，并将自己的solrhome地址加到配置文件中

<env-entry>

<env-entry-name>solr/home</env-entry-name>

<env-entry-value>D:\solr\_home</env-entry-value>

<env-entry-type>java.lang.String</env-entry-type>

</env-entry>

5、启动tomcat8，访问localhost:8080/solr

若出现403的错误，修改solr项目中的web.xml，将169行左右的security-constraint代码注释掉

<!--

<security-constraint>

<web-resource-collection>

<web-resource-name>Disable TRACE</web-resource-name>

<url-pattern>/</url-pattern>

<http-method>TRACE</http-method>

</web-resource-collection>

<auth-constraint/>

</security-constraint>

<security-constraint>

<web-resource-collection>

<web-resource-name>Enable everything but TRACE</web-resource-name>

<url-pattern>/</url-pattern>

<http-method-omission>TRACE</http-method-omission>

</web-resource-collection>

</security-constraint>

-->

若出现404错误，连接地址改为localhost:8080/solr/index.html试试

-----------------------------------------------------------------

**四、配置solr自带的中文分词(和IK的区别是不能自己添加词库)：**

http://www.cnblogs.com/wander1129/archive/2017/04/05/6658828.html

1、复制solr-6.5.0/contrib/analysis-extras/lucene-libs/lucene-analyzers-smartcn-6.5.0.jar到apache-tomcat-8.5.12/webapps/solr/WEB-INF/lib/目录下。

2、为core添加对中文分词的支持。编辑mycore下conf下的managed-schema文件.

在文件的</schema>前添加

<fieldType name="text\_smartcn" class="solr.TextField" positionIncrementGap="0">

<analyzer type="index">

<tokenizer class="org.apache.lucene.analysis.cn.smart.HMMChineseTokenizerFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="org.apache.lucene.analysis.cn.smart.HMMChineseTokenizerFactory"/>

</analyzer></fieldType>

重启tomcat

-----------------------------------------------------------------

**pom.xml**

<!-- solr 搜索 -->

<dependency>

<groupId>org.apache.solr</groupId>

<artifactId>solr-solrj</artifactId>

<version>4.6.0</version>

</dependency>

-----------------------------------------------------------------

**SolrTest.java**

**private** **static** HttpSolrServer server = **null**;

**private** HttpSolrServer getSolrServer() {

**if** (server == **null**) {

server = **new** HttpSolrServer("http://localhost:8081/solr/my\_core");

server.setConnectionTimeout(1000);

server.setDefaultMaxConnectionsPerHost(100);

server.setMaxTotalConnections(100);

}

**return** server;

}

-----------------------------------------------------------------

/\*\*

\* 添加一个Entity到索引库

\*/

@Test

**public** **void** addBean() {

SolrBean solr = **new** SolrBean();

solr.setId("2002");

solr.setContent("hello solr");

solr.setTitle("solr\_article");

**try** {

UpdateResponse response = server.addBean(solr);

server.commit();

logger.info("########## Query Time :" + response.getQTime());

logger.info("########## Elapsed Time :" + response.getElapsedTime());

logger.info("########## Status :" + response.getStatus());

} **catch** (SolrServerException | IOException e) {

logger.error("", e);

}

}

-----------------------------------------------------------------

@Test

**public** PaginationResult<SolrBean> selectBean(String keyword, String articleType, Integer pageNo, Integer countTotal) {

countTotal = countTotal == **null** ? 0 : countTotal;

SolrQuery solrQuery = **new** SolrQuery();

SimplePage page = **new** SimplePage();

solrQuery.set("q", "(title:" + keyword + " OR content:" + keyword + ") AND articleType:" + articleType);

//根据pageNO pageSize 分页查询

**if** (pageNo == **null** || pageNo == 1) {

solrQuery.setStart(0);

solrQuery.setRows(PageSize.SIZE20.getSize());

} **else** {

page = **new** SimplePage(pageNo, countTotal, PageSize.SIZE20.getSize());

solrQuery.setStart(page.getStart());

solrQuery.setRows(page.getEnd());

}

//高亮

solrQuery.setHighlight(**true**);

solrQuery.addHighlightField("title");

solrQuery.setHighlightSimplePre("<span class=\"red\">");

solrQuery.setHighlightSimplePost("</span>");

server = getSolrServer();

QueryResponse response = **null**;

**try** {

response = server.query(solrQuery);

} **catch** (SolrServerException e) {

e.printStackTrace();

}

PaginationResult<SolrBean> result = **new** PaginationResult<SolrBean>();

**if** (**null** != response) {

**int** totalCount = (**int**) response.getResults().getNumFound(); // 设置总数

page.setCountTotal(totalCount);

List<SolrBean> resultList = response.getBeans(SolrBean.**class**); // response返回的结果集封装到SolrBean

Map<String, Map<String, List<String>>> map = response.getHighlighting(); // map中处理后的title（含有span标签） set进resultList

**for** (SolrBean bean : resultList) {

Map<String, List<String>> values = map.get(bean.getId());

**if** (values != **null**) {

List<String> titles = values.get("title");

**if** (titles != **null** && !titles.isEmpty()) {

bean.setTitle(titles.get(0));

}

}

}

result = **new** PaginationResult<SolrBean>(page, resultList);

}

**return** result;

}

--------------------------------------------------------------------------------------------------------------------

# 时间工具

-----------------------------------------------------------------

**DateUtil.java**

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**import** com.ulewo.po.enums.DateTimePatternEnum;

**public** **class** DateUtil {

/\*\*

\* 锁对象

\*/

**private** **static** **final** Object lockObj = **new** Object();

/\*\*

\* 存放不同的日期模板格式的sdf的Map

\*/

**private** **static** Map<String, ThreadLocal<SimpleDateFormat>> sdfMap = **new** HashMap<String, ThreadLocal<SimpleDateFormat>>();

**private** **static** **final** **long** ONE\_MINUTE = 60000L;

**private** **static** **final** **long** ONE\_HOUR = 3600000L;

**private** **static** **final** **long** ONE\_DAY = 86400000L;

**private** **static** **final** **int** AGO\_DAY\_0 = 0, AGO\_DAY\_1 = 1, AGO\_DAY\_2 = 2, AGO\_DAY\_7 = 7;

**private** **static** **final** String JUST\_NOW = "刚刚";

**private** **static** **final** String ONE\_MINUTE\_AGO = "分钟前";

**private** **static** **final** String ONE\_HOUR\_AGO = "小时前";

**private** **static** **final** String ONE\_DAY\_AGO = "天前";

**private** **static** **final** String TWO\_DAY\_AGO = "昨天";

**private** **static** **final** String THTEE\_DAY\_AGO = "前天";

**private** **static** **final** String[] WEEKCN = {"周日", "周一", "周二", "周三", "周四", "周五", "周六"};

/\*\*

\* 返回一个ThreadLocal的sdf,每个线程只会new一次sdf

\* **@param** pattern

\* **@return**

\*/

**private** **static** SimpleDateFormat getSdf(**final** String pattern) {

ThreadLocal<SimpleDateFormat> tl = sdfMap.get(pattern);

// 此处的双重判断和同步是为了防止sdfMap这个单例被多次put重复的sdf

**if** (tl == **null**) {

**synchronized** (lockObj) {

tl = sdfMap.get(pattern);

**if** (tl == **null**) {

// 只有Map中还没有这个pattern的sdf才会生成新的sdf并放入map

// 这里是关键,使用ThreadLocal<SimpleDateFormat>替代原来直接new SimpleDateFormat

tl = **new** ThreadLocal<SimpleDateFormat>() {

@Override

**protected** SimpleDateFormat initialValue() {

**return** **new** SimpleDateFormat(pattern);

}

};

sdfMap.put(pattern, tl);

}

}

}

**return** tl.get();

}

/\*\*

\* 是用ThreadLocal<SimpleDateFormat>来获取SimpleDateFormat,这样每个线程只会有一个SimpleDateFormat

\* **@param** date

\* **@param** pattern

\* **@return**

\*/

**public** **static** String format(Date date, String pattern) {

**return** getSdf(pattern).format(date);

}

**public** **static** Date parse(String dateStr, String pattern) {

**try** {

**return** getSdf(pattern).parse(dateStr);

} **catch** (ParseException e) {

e.printStackTrace();

}

**return** **new** Date();

}

/\*\*

\* getTotalDayAndFirstDay4Month:(获取某年某月第多少天是周几，这个月总共多少天). <br/>

\* **@param** year

\* **@param** month

\* **@param** day

\* **@return**

\* **@since** JDK 1.7

\*/

**public** **static** Map<String, Integer> getTotalDayAndFirstWeekDay4Month(**int** year, **int** month, **int** day) {

Map<String, Integer> result = **new** HashMap<String, Integer>();

// 获取一个月有多少天

Calendar cal = Calendar.getInstance();

cal.set(Calendar.YEAR, year);

cal.set(Calendar.MONTH, month - 1);// Java月份才0开始算 1代表上一个月

**int** dateOfMonth = cal.getActualMaximum(Calendar.DATE);

// 获取当前日期是星期几

cal.set(Calendar.DATE, day);

**int** week = cal.get(Calendar.DAY\_OF\_WEEK);

result.put("totalDay", dateOfMonth);

result.put("firstWeekDay", week);

**return** result;

}

**public** **static** Date getDayYYYYMMDD(Date date) {

String dateStr = format(date, DateTimePatternEnum.YYYY\_MM\_DD.getPattern());

**return** parse(dateStr, DateTimePatternEnum.YYYY\_MM\_DD.getPattern());

}

/\*\*

\* beforeNowDate:(判断时间字符串是否在当天之前). <br/>

\* **@param** date

\* **@return**

\* **@since** JDK 1.7

\*/

**public** **static** **boolean** beforeNowDate(String date) {

**try** {

Date d = getDayYYYYMMDD(**new** Date());

**boolean** flag = **new** SimpleDateFormat(DateTimePatternEnum.YYYY\_MM\_DD.getPattern()).parse(date).before(d);

**return** flag;

} **catch** (ParseException e) {

**return** **false**;

}

}

**public** **static** **int** getWeek(Date date) {

Calendar c = Calendar.getInstance();

c.setTime(date);

**int** w = c.get(Calendar.DAY\_OF\_WEEK) - 1;

**return** w;

}

**public** **static** String getWeekCN(Date date) {

**int** w = getWeek(date);

**return** WEEKCN[w];

}

**public** **static** String friendly\_time(Date sourceDate) {

Date curDate = **new** Date();

Date sourceDateYMD = DateUtil.getDayYYYYMMDD(sourceDate);

Date curDateYMD = DateUtil.getDayYYYYMMDD(curDate);

//几天前

**long** daysAgo = (curDateYMD.getTime() - sourceDateYMD.getTime()) / ONE\_DAY;

//当前

**if** (daysAgo == AGO\_DAY\_0) {

**return** getCurDayInfo(sourceDate, curDate);

} **else** **if** (daysAgo == AGO\_DAY\_1) {

//昨天

**return** TWO\_DAY\_AGO + " " + getHourAndMin(sourceDate);

} **else** **if** (daysAgo == AGO\_DAY\_2) {

//前天

**return** THTEE\_DAY\_AGO + " " + getHourAndMin(sourceDate);

} **else** **if** (daysAgo > AGO\_DAY\_2 && daysAgo <= AGO\_DAY\_7) {

//几天前

**return** (daysAgo - 1) + ONE\_DAY\_AGO;

} **else** {//超过一个星期，不格式化

**return** DateUtil.format(sourceDate, DateTimePatternEnum.YYYY\_MM\_DD.getPattern());

}

}

**private** **static** String getHourAndMin(Date date) {

Calendar c = Calendar.getInstance();

c.setTime(date);

**return** String.format("%02d", c.get(Calendar.HOUR\_OF\_DAY)) + ":" + String.format("%02d", c.get(Calendar.MINUTE));

}

**private** **static** String getCurDayInfo(Date sourceDate, Date curDate) {

**long** secondsAgo = curDate.getTime() - sourceDate.getTime();

**if** (secondsAgo / ONE\_MINUTE <= 0) {//刚刚

**return** JUST\_NOW;

} **else** **if** (secondsAgo / ONE\_MINUTE > 0 && secondsAgo / ONE\_HOUR == 0) {

**return** secondsAgo / ONE\_MINUTE + ONE\_MINUTE\_AGO;

} **else** {

**return** secondsAgo / ONE\_HOUR + ONE\_HOUR\_AGO;

}

}

**public** **static** **boolean** isNew(Date date) {

Calendar c = Calendar.getInstance();

**long** max = c.getTimeInMillis() - date.getTime();

**if** (max / 86400000 <= 1) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** **static** **boolean** valiDateTime(String timeStr) {

**if** (StringTools.isEmpty(timeStr)) {

**return** **false**;

}

String format = "((19|20)[0-9]{2})-(0?[1-9]|1[012])-(0?[1-9]|[12][0-9]|3[01]) " + "([01]?[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]";

Pattern pattern = Pattern.compile(format);

Matcher matcher = pattern.matcher(timeStr);

**if** (matcher.matches()) {

pattern = Pattern.compile("(\\d{4})-(\\d+)-(\\d+).\*");

matcher = pattern.matcher(timeStr);

**if** (matcher.matches()) {

**int** y = Integer.valueOf(matcher.group(1));

**int** m = Integer.valueOf(matcher.group(2));

**int** d = Integer.valueOf(matcher.group(3));

**if** (d > 28) {

Calendar c = Calendar.getInstance();

c.set(y, m - 1, 1);

**int** lastDay = c.getActualMaximum(Calendar.DAY\_OF\_MONTH);

**return** (lastDay >= d);

}

}

**return** **true**;

}

**return** **false**;

}

**public** **static** **boolean** isValidDate(String sDate) {

String datePattern1 = "\\d{4}-\\d{2}-\\d{2}";

String datePattern2 = "^((\\d{2}(([02468][048])|([13579][26]))" + "[\\-\\/\\s]?((((0?[13578])|(1[02]))[\\-\\/\\s]?((0?[1-9])|([1-2][0-9])|" + "(3[01])))|(((0?[469])|(11))[\\-\\/\\s]?((0?[1-9])|([1-2][0-9])|(30)))|(0?2[\\-\\/\\s]?" + "((0?[1-9])|([1-2][0-9])))))|(\\d{2}(([02468][1235679])|([13579][01345789]))[\\-\\/\\s]?(" + "(((0?[13578])|(1[02]))[\\-\\/\\s]?((0?[1-9])|([1-2][0-9])|(3[01])))|(((0?[469])|(11))[\\-\\/\\s]?" + "((0?[1-9])|([1-2][0-9])|(30)))|(0?2[\\-\\/\\s]?((0?[1-9])|(1[0-9])|(2[0-8]))))))";

**if** (!StringTools.isEmpty(sDate)) {

Pattern pattern = Pattern.compile(datePattern1);

Matcher match = pattern.matcher(sDate);

**if** (match.matches()) {

pattern = Pattern.compile(datePattern2);

match = pattern.matcher(sDate);

**return** match.matches();

} **else** {

**return** **false**;

}

}

**return** **false**;

}

**public** **static** **void** main(String[] args) **throws** ParseException {

String str = "2015-12-15 16:30:30";

Date date = parse(str, DateTimePatternEnum.YYYY\_MM\_DD\_HH\_MM\_SS.getPattern());

System.out.println(friendly\_time(date));

}

**public** **static** String getNextMonthDay() {

Calendar calender = Calendar.getInstance();

calender.add(Calendar.MONTH, 1);

Date date = calender.getTime();

**return** DateUtil.format(date, DateTimePatternEnum.YYYY\_MM\_DD.getPattern());

}

}

-----------------------------------------------------------------

**DateTimePatternEnum.java**

**public** **enum** DateTimePatternEnum {

YYYY\_MM\_DD\_HH\_MM\_SS("yyyy-MM-dd HH:mm:ss"), YYYYMM("yyyyMM"), YYYYMMDD("yyyyMMdd"), YYYY("YYYY"), MM\_POINT\_DD("MM.dd"), YYYY\_MM\_DD

("yyyy-MM-dd");

**private** String pattern;

**private** DateTimePatternEnum(String pattern) {

**this**.pattern = pattern;

}

**public** String getPattern() {

**return** pattern;

}

**public** **void** setPattern(String pattern) {

**this**.pattern = pattern;

}

}

--------------------------------------------------------------------------------------------------------------------

# Http状态码

-----------------------------------------------------------------

**ResponseCode.java**

@JsonFormat(shape = JsonFormat.Shape.OBJECT)

**public** **enum** ResponseCode {

/\*\*

\* code:200<br>

\* 请求成功

\*/

SUCCESS(200, "请求成功"),

/\*\*

\* code:403<br>

\* 没有权限

\*/

NOPERMISSION(403, "没有权限"),

/\*\*

\* code:415<br>

\* 业务异常

\*/

BUSINESSERROR(415, "业务异常"),

/\*\*

\* code:425<br>

\* 验证码错误

\*/

CODEERROR(425, "验证码错误"),

/\*\*

\* code:500<br>

\* 服务器错误

\*/

SERVERERROR(500, "服务器错误"),

/\*\*

\* code:401<br>

\* 登录超时

\*/

LOGINTIMEOUT(401, "登录超时"),

/\*\*

\* 登陆错误次数超过限制

\*/

MOREMAXLOGINCOUNT(405, "登陆重试次数超过3次");

**private** **int** code;

**private** String desc;

**private** ResponseCode(**int** code, String desc) {

**this**.code = code;

**this**.desc = desc;

}

**public** **int** getCode() {

**return** code;

}

**public** **void** setCode(**int** code) {

**this**.code = code;

}

**public** String getDesc() {

**return** desc;

}

**public** **void** setDesc(String desc) {

**this**.desc = desc;

}

**public** **static** ResponseCode getResponseByCode(Integer code) {

**for** (ResponseCode at : ResponseCode.values()) {

**if** (at.code == code) {

**return** at;

}

}

**return** **null**;

}

}

--------------------------------------------------------------------------------------------------------------------

# chrome手机模式下小圆点消失

-----------------------------------------------------------------

HDMI线下部分显卡驱动的造成的,更改设置

以Intel核显为例，在显卡控制面板里选择「显示-量化范围」(显示器>一般设置>高级)（在nvidia那边叫动态范围），里面有三项：

默认范围 Default Range  
有限范围 Limited Range  
全范围 Full Range

勾上「全范围」，保存，看看chrome里那个小圆点是不是出来了？

--------------------------------------------------------------------------------------------------------------------

# 文件上传、删除、下载

-----------------------------------------------------------------

**upload.html**

<form action=*"/fileUpload"* enctype=*"multipart/form-data"* method=*"post"*>

<input type=*"file"* name=*"uploadfile"*/>

<button type=*"submit"* value=*"1"*>提交</button>

</form>

-----------------------------------------------------------------

**web.xml**

<servlet>

<servlet-name>springMVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath\*:dispatcher-servlet.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springMVC</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

-----------------------------------------------------------------

**Dispatcher-servlet.xml**

<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<property name=*"maxUploadSize"* value=*"100000"*/>

<property name=*"maxInMemorySize"* value=*"10240"* />

</bean>

-----------------------------------------------------------------

**FileUploadAction.java**

@Controller

**public** **class** FileUploadAction {

**private** **final** **static** **int** MAX\_FILE = 1024 \* 1024 \* 3;

**private** Logger log = LoggerFactory.getLogger(FileUploadAction.**class**);

@ResponseBody

@RequestMapping(value = "/fileUpload")

**public** Map<String, Object> fileupload(HttpSession session, MultipartHttpServletRequest request, HttpServletResponse response) {

Map<String, Object> map = **new** HashMap<String, Object>();

**try** {

String realPath = "D:/file";

Iterator<String> itr = request.getFileNames();

**if** (itr.hasNext()) {

MultipartFile multipartFile = request.getFile(itr.next());

**long** size = multipartFile.getSize();

**if** (size > MAX\_FILE) {

map.put("responseCode", ResponseCode.BUSINESSERROR.getCode());

map.put("msg", "文件最大不能超过3M");

**return** map;

}

String fileName = multipartFile.getOriginalFilename();

String suffix = fileName.substring(fileName.lastIndexOf(".") + 1);

**if** (!"zip".equalsIgnoreCase(suffix) && !"rar".equalsIgnoreCase(suffix)) { //对文件后缀的过滤

map.put("responseCode", ResponseCode.BUSINESSERROR.getCode());

map.put("msg", "文件类型只能是压缩包");

**return** map;

}

String current = String.valueOf(System.currentTimeMillis());

fileName = current + "." + suffix;

String saveDir = DateUtil.format(**new** Date(), DateTimePatternEnum.YYYYMM.getPattern());

String savePath = saveDir + "/" + fileName;

String fileDir = "D:/file/temp/" + saveDir;

File dir = **new** File(fileDir);

**if** (!dir.exists()) {

dir.mkdirs();

}

String filePath = fileDir + "/" + fileName;

File file = **new** File(filePath);

multipartFile.transferTo(file);

map.put("responseCode", ResponseCode.SUCCESS.getCode());

map.put("savePath", savePath);

**return** map;

} **else** {

map.put("responseCode", ResponseCode.BUSINESSERROR.getCode());

map.put("msg", "上传文件未找到");

**return** map;

}

} **catch** (Exception e) {

map.put("responseCode", ResponseCode.BUSINESSERROR.getCode());

map.put("msg", "服务器异常，上传失败");

log.error(e.getMessage(), e);

**return** map;

}

}

@ResponseBody

@RequestMapping(value = "/fileDelete", method = RequestMethod.POST)

**public** Map<String, Object> deleteFile(HttpSession session, HttpServletRequest request, String fileName) {

Map<String, Object> modelMap = **new** HashMap<String, Object>();

**try** {

String realPath = "D:/file/temp";

File file = **new** File(realPath + fileName);

**if** (file.exists()) {

file.delete();

}

modelMap.put("result", ResponseCode.SUCCESS.getCode());

**return** modelMap;

} **catch** (Exception e) {

log.error("删除附件异常" + e.getMessage(), e);

modelMap.put("result", ResponseCode.SERVERERROR.getCode());

**return** modelMap;

}

}

}

@RequestMapping(value = "download")

**public** ModelAndView downloadAttachment(HttpSession session, HttpServletRequest request, HttpServletResponse response, Integer attachmentId) {

InputStream in = **null**;

OutputStream out = **null**;

**try** {

String filePath = "D:/file/temp/201706/1496742228384.rar";

File file = **new** File(filePath);

in = **new** FileInputStream(file);

out = response.getOutputStream();

response.setContentType("application/x-msdownload; charset=UTF-8");

String fileName = "1496742228384.rar";

// 解决中文文件名乱码问题

**if** (request.getHeader("User-Agent").toLowerCase().indexOf("firefox") > 0) {

fileName = **new** String(fileName.getBytes("UTF-8"), "ISO8859-1"); // firefox浏览器

} **else** {

fileName = URLEncoder.encode(fileName, "UTF-8");// IE浏览器

}

response.setHeader("Content-Disposition", "attachment;filename=\"" + fileName + "\"");

**byte**[] byteData = **new** **byte**[1024 \* 5];

**int** len = 0;

**while** ((len = in.read(byteData)) != -1) {

out.write(byteData, 0, len); // write

}

out.flush();

**return** **null**;

} **catch** (Exception e) {

System.out.println("下载异常");

} **finally** {

**try** {

**if** (in != **null**) {

in.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

**try** {

**if** (out != **null**) {

out.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** **null**;

}

--------------------------------------------------------------------------------------------------------------------

# Quartz任务调度（CronTrigger）

-----------------------------------------------------------------

**pom.xml**

<!-- 定时任务 -->

<dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz</artifactId>

<version>2.1.1</version>

<exclusions>

<exclusion>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

</exclusion>

<exclusion>

<artifactId>slf4j-api</artifactId>

<groupId>org.slf4j</groupId>

</exclusion>

</exclusions>

</dependency>

-----------------------------------------------------------------

**StaticsTask.java（处理具体业务的Task）**

**public** **class** StaticsTask {

**public** **void** staticsInfo() {

System.out.println("This is staticsInfo method!");

}

**public** **void** noticeSignInTask() {

System.out.println("This is noticeSignInTask method!");

}

}

-----------------------------------------------------------------

**TaskMessage.java**

**public** **class** TaskMessage {

**private** Integer id;

**private** String taskClassz;

**private** String taskMethod;

**private** String taskTime;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getTaskClassz() {

**return** taskClassz;

}

**public** **void** setTaskClassz(String taskClassz) {

**this**.taskClassz = taskClassz;

}

**public** String getTaskMethod() {

**return** taskMethod;

}

**public** **void** setTaskMethod(String taskMethod) {

**this**.taskMethod = taskMethod;

}

**public** String getTaskTime() {

**return** taskTime;

}

**public** **void** setTaskTime(String taskTime) {

**this**.taskTime = taskTime;

}

}

-----------------------------------------------------------------

**CronTriggerRunner.java**

@Component

**public** **class** CronTriggerRunner {

**private** **final** String TASKNAME = "task\_";

**private** **final** String TASKGROUP = "task\_gourp\_";

/\*\*

\* 保存调度

\* runJob:(这里用一句话描述这个方法的作用)

\* **@param** task

\* **@exception** SchedulerException

\* **@since** JDK 1.7

\*/

**public** **void** saveJob(TaskMessage task, **boolean** immediateExecution) **throws** SchedulerException {

SchedulerFactory schedulerFactory = **new** StdSchedulerFactory();

Scheduler scheduler = schedulerFactory.getScheduler();

TriggerKey triggerKey = getTriggerKey(task);

CronTrigger trigger = (CronTrigger) scheduler.getTrigger(triggerKey);

// 不存在，创建一个

**if** (**null** == trigger) {

JobDetail jobDetail = JobBuilder.newJob(DefaultJob.**class**).withIdentity(TASKNAME + task.getId(), TASKGROUP + task.getId()).build();

jobDetail.getJobDataMap().put("task\_message", task);

// 表达式调度构建器

CronScheduleBuilder scheduleBuilder = CronScheduleBuilder.cronSchedule(task.getTaskTime());

// 按新的表达式构建一个新的trigger

trigger = TriggerBuilder.newTrigger().withIdentity(TASKNAME + task.getId(), TASKGROUP + task.getId()).withSchedule(scheduleBuilder).build();

scheduler.scheduleJob(jobDetail, trigger);

scheduler.start();

} **else** {

// trigger已存在，则更新相应的定时设置

CronScheduleBuilder scheduleBuilder = CronScheduleBuilder.cronSchedule(task.getTaskTime());

// 按新的cronExpression表达式重新构建trigger

trigger = trigger.getTriggerBuilder().withIdentity(triggerKey).withSchedule(scheduleBuilder).build();

// 按新的trigger重新设置job执行

scheduler.rescheduleJob(triggerKey, trigger);

}

**if** (immediateExecution) {

**this**.triggerJob(task, scheduler);

}

}

/\*\*

\* 暂停

\* pauseJob:(这里用一句话描述这个方法的作用). <br/>

\* **@exception** SchedulerException

\* **@since** JDK 1.7

\*/

**public** **void** pauseJob(TaskMessage task) **throws** SchedulerException {

SchedulerFactory schedulerFactory = **new** StdSchedulerFactory();

Scheduler scheduler = schedulerFactory.getScheduler();

JobKey jobKey = getJobKey(task);

scheduler.pauseJob(jobKey);

}

/\*\*

\* 删除job

\* delJob:(这里用一句话描述这个方法的作用). <br/>

\* **@param** task

\* **@exception** SchedulerException

\* **@since** JDK 1.7

\*/

**public** **void** delJob(TaskMessage task) **throws** SchedulerException {

SchedulerFactory schedulerFactory = **new** StdSchedulerFactory();

Scheduler scheduler = schedulerFactory.getScheduler();

JobKey jobKey = getJobKey(task);

scheduler.deleteJob(jobKey);

}

/\*\*

\* 立即执行

\* triggerJob:(这里用一句话描述这个方法的作用). <br/>

\* **@param** task

\* **@exception** SchedulerException

\* **@since** JDK 1.7

\*/

**public** **void** triggerJob(TaskMessage task, Scheduler scheduler) **throws** SchedulerException {

JobKey jobKey = getJobKey(task);

scheduler.triggerJob(jobKey);

}

**private** JobKey getJobKey(TaskMessage task) {

JobKey jobKey = JobKey.jobKey(TASKNAME + task.getId(), TASKGROUP + task.getId());

**return** jobKey;

}

**private** TriggerKey getTriggerKey(TaskMessage task) {

TriggerKey triggerKey = TriggerKey.triggerKey(TASKNAME + task.getId(), TASKGROUP + task.getId());

**return** triggerKey;

}

}

-----------------------------------------------------------------

**TaskTest.java**

@ContextConfiguration(locations = {

"classpath:applicationContext.xml","classpath:dispatcher-servlet.xml"})

**public** **class** TaskTest **extends** AbstractJUnit4SpringContextTests {

@Autowired

**private** CronTriggerRunner cronTriggerRunner;

@Test

**public** **void** saveJob() **throws** SchedulerException {

Task task = **new** Task();

task.setId(6);

task.setTaskClassz("example.task.StaticsTask");

task.setTaskMethod("staticsInfo");

task.setTaskTime("0 55 23 \* \* ? \*");

task.setLastupdateTime(DateUtil.parse("2015-12-19 18:24:43", DateTimePatternEnum.YYYY\_MM\_DD\_HH\_MM\_SS.getPattern()));

task.setTaskStatus(Taskstatus.NORMAL);

task.setDescription("统计数量");

TaskMessage taskMessage = convertTask2TaskMessage(task);

cronTriggerRunner.saveJob(taskMessage, **true**);

}

**private** TaskMessage convertTask2TaskMessage(Task task) {

TaskMessage taskMessage = **new** TaskMessage();

taskMessage.setId(task.getId());

taskMessage.setTaskClassz(task.getTaskClassz());

taskMessage.setTaskMethod(task.getTaskMethod());

taskMessage.setTaskTime(task.getTaskTime());

**return** taskMessage;

}

}

--------------------------------------------------------------------------------------------------------------------

# Logback配置及使用

-----------------------------------------------------------------

**pom.xml**

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.1.3</version>

</dependency>

-----------------------------------------------------------------

**Logback.xml（放进resources文件夹下即可，不需要读入）**

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<configuration scan=*"true"* scanPeriod=*"10 minutes"*>

<property name=*"LOG\_HOME"* value=*"D:/logs"*/>

<property name=*"log.maxHistory"* value=*"90"* />

<property name=*"log.pattern"* value=*"%d{yyyy-MM-dd HH:mm:ss} [%p][%c][%M][%L]-> %m%n"*/>

<appender name=*"stdout"* class=*"ch.qos.logback.core.ConsoleAppender"*>

<layout class=*"ch.qos.logback.classic.PatternLayout"*>

<pattern>%d{yyyy-MM-dd HH:mm:ss} [%p][%c][%M][%L]-> %m%n</pattern>

</layout>

</appender>

<!-- DEBUG -->

<appender name=*"debugAppender"* class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>

<!-- 当前日志输出文件名 -->

<file>${LOG\_HOME}/debug.log</file>

<rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>

<!-- 文件名称 -->

<fileNamePattern>${LOG\_HOME}/debug/debug.%d{yyyy-MM-dd}.%i.log

</fileNamePattern>

<!-- 文件最大保存天数 -->

<MaxHistory>30</MaxHistory>

<!--当前日志文件满了5KB以后会拆分进debug文件夹下的debug.%d{yyyy-MM-dd}.%i.log文件，i从0开始，标志拆分了几个文件-->

<TimeBasedFileNamingAndTriggeringPolicy class=*"ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"*>

<MaxFileSize>5KB</MaxFileSize>

</TimeBasedFileNamingAndTriggeringPolicy>

</rollingPolicy>

<encoder>

<pattern>${log.pattern}</pattern>

</encoder>

<!--一个文件夹存放一种级别日志，过滤，只需要DEBUG日志-->

<filter class=*"ch.qos.logback.classic.filter.LevelFilter"*>

<level>DEBUG</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

</appender>

<!-- INFO -->

<appender name=*"infoAppender"* class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>

<!-- 文件路径 -->

<file>${LOG\_HOME}/debug.log</file>

<rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>

<!-- 文件名称 -->

<fileNamePattern>${LOG\_HOME}/debug/debug.%d{yyyy-MM-dd}.%i.log

</fileNamePattern>

<!-- 文件最大保存天数 -->

<MaxHistory>30</MaxHistory>

<TimeBasedFileNamingAndTriggeringPolicy class=*"ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"*>

<MaxFileSize>5KB</MaxFileSize>

</TimeBasedFileNamingAndTriggeringPolicy>

</rollingPolicy>

<encoder>

<pattern>${log.pattern}</pattern>

</encoder>

<filter class=*"ch.qos.logback.classic.filter.LevelFilter"*>

<level>INFO</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

</appender>

<!-- ERROR -->

<appender name=*"errorAppender"* class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>

<!-- 文件路径 -->

<file>${LOG\_HOME}/debug.log</file>

<rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>

<!-- 文件名称 -->

<fileNamePattern>${LOG\_HOME}/debug/debug.%d{yyyy-MM-dd}.%i.log

</fileNamePattern>

<!-- 文件最大保存天数 -->

<MaxHistory>30</MaxHistory>

<TimeBasedFileNamingAndTriggeringPolicy class=*"ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"*>

<MaxFileSize>5KB</MaxFileSize>

</TimeBasedFileNamingAndTriggeringPolicy>

</rollingPolicy>

<encoder>

<pattern>${log.pattern}</pattern>

</encoder>

<filter class=*"ch.qos.logback.classic.filter.LevelFilter"*>

<level>ERROR</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

</appender>

<logger name=*"wf"* level=*"INFO"*/>

<root level=*"INFO"*>

<appender-ref ref=*"stdout"*/>

<appender-ref ref=*"debugAppender"*/>

<appender-ref ref=*"infoAppender"* />

<appender-ref ref=*"errorAppender"* />

</root>

</configuration>

-----------------------------------------------------------------

**LogbackDemo.java**

**public** **class** LogbackDemo {

**private** **static** Logger log = LoggerFactory.getLogger(LogbackDemo.**class**);

**public** **static** **void** main(String args[]) {

System.out.println("--------------");

log.trace("======trace");

log.debug("======debug");

log.info("======info");

log.warn("======warn");

log.error("======error");

System.out.println("--------------");

}

}

--------------------------------------------------------------------------------------------------------------------